KYoT: Self-sovereign IoT Identification with a Physically Unclonable Function

Niya, Sina Rafati ; Jeffrey, Benjamin ; Stiller, Burkhard

KYoT: Self-sovereign IoT Identification with a Physically Unclonable Function

Sina Rafati Niya, Benjamin Jeffrey, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH
Binzmühlestrasse 14, CH—8050 Zürich, Switzerland
Emails: [rafati|stiller@ifi.uzh.ch], benjamin.jeffrey@uzh.ch

***Abstract*—The integration of Internet-of-Things (IoT) and Blockchains (BC) for trusted and decentralized approaches enabled modern use cases, such as supply chain tracing, smart cities, and IoT data marketplaces. For these it is essential to identify reliably IoT devices, since the producer-consumer trust is not guaranteed by a Trusted Third Party (TTP). Therefore, this work proposes a Know Your IoT device platform (KYoT), which enables the self-sovereign identification of IoT devices on the Ethereum BC. KYoT permits manufacturers and device owners to register and verify IoT devices in a self-sovereign fashion, while data storage security is ensured. KYoT deploys an SRAM-based (Static Random Access Memory) Physically Unclonable Function (PUF), which takes advantage of the manufacturing variability of devices' SRAM chips to derive a unique identifying key for each IoT device. The self-sovereign identification mechanism introduced is based on the ERC 734 and ERC 735 Ethereum identity standards.**

***Index Terms*—Self-sovereign Identification, Physically Unclonable Function PUF, Ethereum, Blockchains, Smart Contract**

I. Introduction

For many IoT use cases data security is key. Therefore, IoT data transmission security is required, where a typical solution encrypts data with a secret key, which is often preserved in a non-volatile storage. However, this is susceptible to possible attacks, where an unauthorized retrieval of the secret key could occur [17]. To overcome such vulnerabilities, this paper exploits IoT devices' hardware characteristics, which are physically unrepeatable and represent a unique construction of hardware as well as its material used. Thus, such an identification of IoT devices can ensure the origin of data.

Proposed methods exist utilizing hardware's unique characteristics for identification, *i.e.*, Physically Unclonable Functions (PUF) [7]. PUFs study on small variations in the characteristics of Integrated Circuits (IC) of IoT devices to derive unique keys based on the individual characteristics of this IC, thus, enabling secret key generation without the need for storage in a non-volatile memory [12]. This is due to the fact that the PUF-generated key can be generated on the fly, while the PUF output will remain the same. The PUF-generated key acts like a hardware's DNA. PUF-based solutions take IC characteristics as input and produce a key that is unique to that IC given. A valid and reliable PUF produces the same key for the same IC with a high probability and must, within reason, produce a different key for each new IC.

While a large number of IoT hardwares exist, this work utilizes Arduino (*e.g.*, Arduino Mega2560 or Arduino Uno) [1] devices. Three pools of memory are used in the microcontroller on AVR-based Arduino boards [4], *i.e., (i)* flash memory (program space), storing the Arduino sketch, *(ii)* SRAM (Static Random Access Memory) for the sketch to create and manipulate variables at run-time, and *(iii)* EEPROM, memory to store long-term information. The SRAM is volatile and data is lost when the power is turned off. The ATmega2560 is the hardware used in this work has an SRAM of 8 kByte [4].

The manufacturing variability of SRAMs causes small mismatches between N-channel and P-channel transistors, which results in individual SRAM cells having a bias toward 0 or 1, upon powering on. Since these cells' values are mostly consistent for each power cycle, they determine the physical characteristic for that chip. Therefore, the start-up values of SRAM bits are used as input for a PUF [17]. Utilizing this attribute with the Arduino SRAM leads to the design and implementation of the Know Your IoT device platform (KYoT) for the identification of Arduino-based IoT devices.

KYoT aims to enforce a reliable approach for self-sovereign identity management based on the Ethereum Blockchain (BC) identity standards [6], [20], [21]. This approach ensures each device is represented on the BC by a Smart Contract (SC). Since SCs are autonomous distributed applications running on Ethereum network, they manage a set of peers based on predefined rules.

The remainder of this paper is organized as follows. Section II presents background and related work followed by the design and implementation description of KYoT in Section III and IV. Sections V and VI discuss the KYoT's approach and contain a brief summary.

II. Background and Related Work

The design and implementation of KYoT involves a range of areas, especially IoT, BCs, self-sovereign identity management, SCs, identity standards of the Ethereum BC.

A. Background

BCs operate on the basis of *(a)* a Peer-to-Peer (P2P) network protocol, *(b)* a copy of the distributed ledger per node, and *(c)* minors, validators, and BC clients providing an immutable, distributed data storage via *consensus mechanisms* deployed. The consensus mechanism is a decentralized and trusted *principle-set* providing the global agreement. An example is the Proof-of-Work (PoW) [8].

Physically Unclonable Functions (PUF) were initiated as “Silicon Physical Random" Functions [12]. Different implementations of circuits identified authenticated individual ICs using Field Programmable Gate Arrays (FPGAs). [14] proposed the use of the power-up state of SRAMs as a fingerprint. [17] investigated this approach as a PUF for a chip identification. The manufacturing process of SRAMs [17] showed that the ATMega 1284p microcontroller contains sufficient entropy to uniquely identify each chip based on its power-up state. Initially, error rates *i.e.,* the number of SRAM cells behaving fully random of SRAM PUFs, are too high to be used for applications such as key generation and authentication. Without utilizing an error correction approach, the PUF output would be useless or hard to be regenerated equally. One such an approach is using a ”Fuzzy Extractor" explained as follows.

Fuzzy Extractors were proposed as a primitive to extract nearly uniform randomness from noisy data in order to securely authenticate biometric data [10]. The primitive can be applied to information that is not reproducible precisely and is not distributed uniformly. The output of fuzzy extractors is error-tolerant, thus, the same output is produced for input data that is slightly variant, as long as the differences stay within a margin.

Ethereum Identity Standards exist to define the identity of users and devices. ERC 734 describes standard functions for uniquely identifiable proxy SCs that can be used by other accounts or SCs. These SCs describe "anything", such as groups of individuals or devices, and act as an identity proxy on the BC. The described identity SC has a key-storage controlling the degree to which other parties can interact with the contract. *E.g.,* upon creation of the contract, a management key is added to the key-storage based on the creator’s account address to ensure that certain functionality can only be executed by the SC creator. The contract also features an execute function to run arbitrary contract calls, which acts as a proxy for whatever instance it is representing [6]. ERC 735 extends these functions to add and remove claims, which are hashed and signed by an identity contract of a trusted third party. In order to issue claims, this contract must first add a claim key to its key-storage, as it will be used to sign the claim and will be later checked by anyone wishing to check, if the claim is valid [20].

B. BC and PUF Solutions for IoT

Major state-of-the-art BC and PUF-based solutions for IoT identification are elaborated in [18]. While the comparison of related work and KYoT (*cf.* Table I) shows major differences, PUFs used to identify IoT devices were introduced by [9] via an identity-based cryptosystem to enable a secure authentication and message encryption between IoT devices. IoT devices run an enrolment procedure, where each node saves its PUF-derived Challenge Response Pair (CRP) in a database hosted by a server node in the IoT network. Upon a communication demand, the server node authenticates them by checking the newly generated PUF data matching the CRPs stored.

[22] proposed a concept to uniquely link physical devices to logical addresses on the BC in order to protect IoT transactions by using Identification Random Access Memory (IDRAM) as a replacement chip for the RAM built into devices to facilitate physical chip identification. [13] utilizes for an authentication scheme SRAM PUFs to generate digital fingerprints, deploying public and private BCs, thus, manufacturers register devices before selling them. [15] combines PUFs with Ethereum SCs to ensure data provenance and data integrity in IoT environments. Authentications are handled by SCs deployed acting as trusted servers. Devices publish the CRP from their PUF implementation to this contract to register themselves. Whenever authentication is necessary, transactions are due by the device, the server contract initiates the verification process.

TABLE I
OVERVIEW OF RELATED WORK

Approach	PUF	BC	Use Case’s Focus
[9]	Not specified	None	Authentication Message encryption
[22]	None (IDRAM)	Not specified	Authentication Message encryption
[13]	SRAM PUF	Not specified	Periodic Identification
[15]	Not specified	Ethereum	Authentication Message encryption
[16]	Various	Ethereum	Device Integrity within supply chain
KYoT	SRAM PUF	Ethereum ERC 734/735	Identification

III. KYOT DESIGN

KYoT’s design is multidimensional. It maps devices to users hence, it includes a Know-Your-Costumer (KYC), a Know-Your-Device (KYD), a PUF algorithm, and smart contracts all integrated in one package. The following presents these dimensions by elaborating on the design details of each.

A. Device Verification:

The process of verifying a device on the KYoT server is comprised of two stages: *(a)* registering devices before a user gets access to a device – done by the manufacturer – and *(b)* a device buyer and owner registering of a device on KYoT. This step requires a unique identifier already received with the purchase of a device. Once successfully registered, the user can proceed to verify her/his device (*cf.* Figure 1).

As a verified presence of an IoT device on the Ethereum BC is key, interactions with other accounts and SCs on the network show parties trusting the device’s identity. To ensure that the device is linked to a device owner with a verified presence on the BC, KYoT deploys Ethereum SCs handling identities based on ERC 734 combined with ERC 735 [2], [3].

B. KYC Claim

Figure 2 shows the steps required to deploy and set-up all SCs, including the user’s verification: The user deploys an identity SC to the Ethereum BC. The required management key is automatically added upon the creation of the SC. This proxy SC requires the user to verify his identity before adding

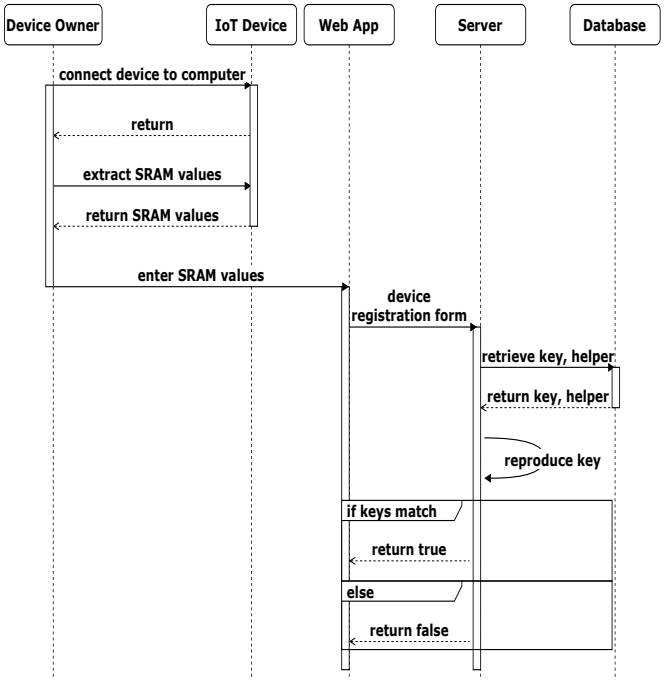

Fig. 1. KYoT Design — Device Verification Processes

devices. KYoT implements the user's registration (via KYC) and the device registration (via KYD) processes. The user has to send the data required to be verified to the KYC service, which will process this verification request. This claim is added to the user's deployed identity SC, at which point the user's identity will be considered as "verified".

If the user chooses to add a device, the user must add a claim key to the identity SC, as this will be required to sign ownership claims for devices, effectively linking devices to the device owner. Subsequently, the user deploys an identity SC for this device, signs an ownership claim, and adds it to the newly created proxy SC. At this point, KYoT has successfully added proxy instances of the user and all devices to the BC, as well as verified the user's identity.

C. KYD Claim

To identify devices added (*cf.* Figure 3) the device must run the verification process. If the KYD platform deems the device to be valid, its proxy SC signs the KYD claim. As with other claims, the device's identity SC adds this claim and shows that it was verified by the KYD. If other SCs or accounts on the BC want to ensure that the identity of this device is verified, they can check, if its identity SC includes a signed KYD claim, which also provides information on the issuer.

IV. Implementation

Specification of KYoT implementation considering the main steps users have to follow for a successful interaction with KYoT's KYD and KYC processes are presented as follows covering the algorithms presented in Section III.

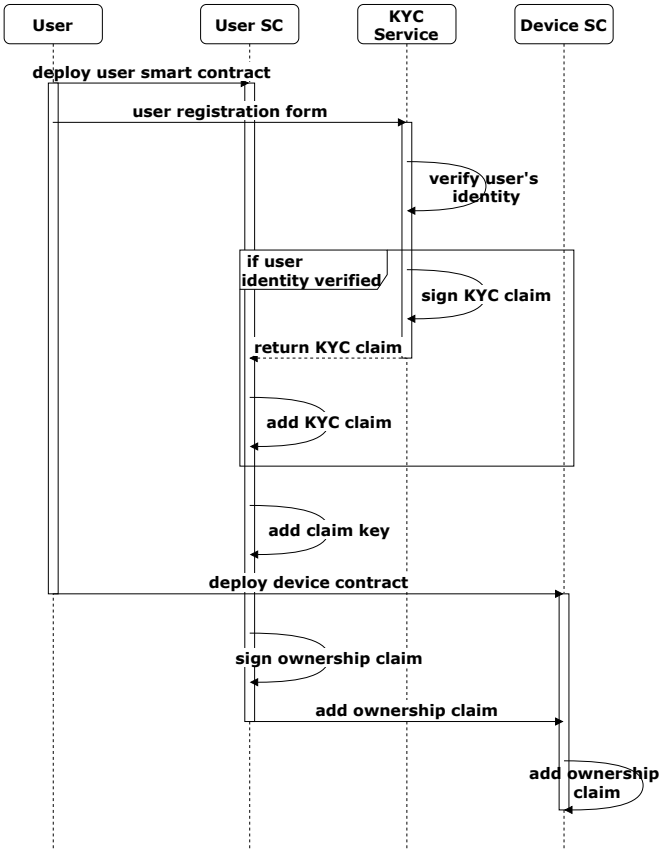

Fig. 2. Deployment and Set-Up of Identity SCs for a User/Device

A. User Registration

Upon a user sign-in for the first time, the registration process is needed and includes interactions with the Web Application (App), the BC, and the KYD server. The interaction with the Ethereum BC is handled by the KYoT's Web App. KYoT's Web App uses the provider injected by the MetaMask [5] extension to interact with the BC. This is done on a user's behalf and has its separate identity SC for the KYD signed by the KYC and KYD claims. Assuming KYoT users are primarily connected to their Ethereum account using the MetaMask browser extension, the sign-in page automatically launches a MetaMask pop-up, where the user can sign in and subsequently approve account access. The next step requires the user to fill in a form displayed on the user registration page. It includes all fields necessary for the user creation request.

```
1  // Deploys an identity smart contract from accounts[0].
2  const result = await new this.web3.eth.Contract(abi)
3    .deploy({ data: bytecode })
4    .send({ from: accounts[0], gas: 3000000 });
5  // Adds a claim key to the deployed SC.
6  await result.methods.addKey(
7    this.web3.utils.keccak256(result.options.address), 3,
         1
8  ).send({ from: accounts[0], gas: 3000000 });
```

Listing 1. SC Deployment for a User Within the Web App's User Service

The next step deploys a proxy SC representing the user on the Ethereum BC. To do this, the form data gathered

in the previous step is passed to the *register* function of the `user.service` in the Web application. The injected Web3 instance is used to deploy a plain version of the `Identity.sol` SC to the BC (*cf.* Listing 1). The second block of code covers the adding of a claim key to the newly created SC. The key is derived from the SC address using a hash function. This key is required later for the issuing of ownership claims to associated devices.

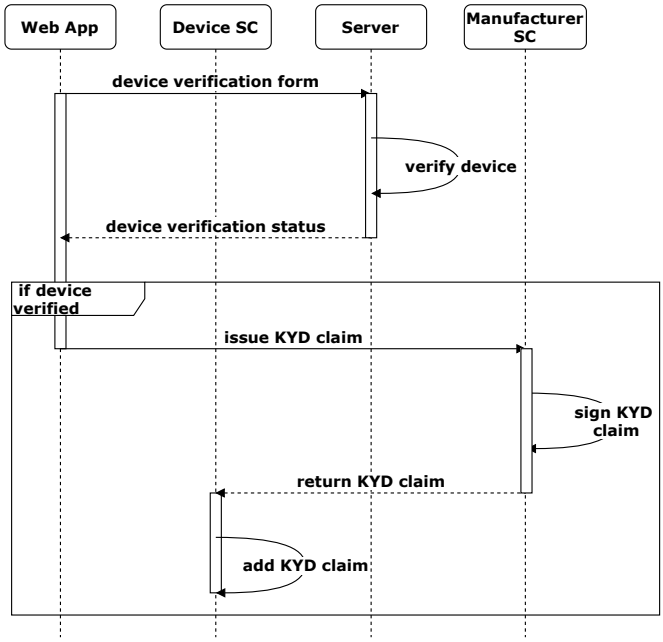

Fig. 3. Issuing a KYD Claim in KYoT

Once these interactions with the BC are approved from the account owner via Metamask, the proxy SC is deployed for the user on the BC and it is ready for adding devices. As one of the benefits of linking devices to a user is accountability, the user needs to be a verified entity that could be held accountable for wrongdoings. Thus, a KYC claim is added to the SC, if the user successfully identifies himself. This process is implemented by filling out a registration form [2], [3].

Listing 2 shows the signing of the KYC claim and adding it to the user's identity SC. The first block of code creates data included in the claim. Due to privacy reasons, KYoT only includes the SC's address. This data is hashed alongside the identity SC address and the designation of a KYC claim ($claimType = 7$). Finally, the hashed data is signed by the KYoT. The signature is subsequently added to a *Claim* object alongside the issuer's Ethereum address and URL. The entire *addClaim* method is encoded into an Application Binary Interface (ABI), such that the identity SC can execute the method. This represents the user by a proxy SC on the BC.

B. Device Registration

In order to register a device, KYoT extracts SRAM data from the device, since it is used to construct a PUF for the device's verification. Devices are registered in the KYoT's database by the platform itself before the device owner receives her/his device. Next, s/he can register it on KYoT using an identification number that was included with the device.

```
// Creates a signature from the contract's address.
const data = this.web3.utils.asciiToHex(
  [result.options.address, 'verified']
    .join(' ')
);
const hashedData = this.web3.utils.soliditySha3(
  result.options.address, 7, data
);
const signature = await kydWeb3.eth.sign(
  hashedData, environment.walletAddress
);
// Creates a KYC claim from the signature.
const addClaimABI = result.methods.addClaim(
  7, 1, environment.contractAddress,
  signature, data, 'https://www.arduino.cc/'
).encodeABI();
// Adds the KYC claim to the identity smart contract.
await result.methods.execute(
  result.options.address, 0, addClaimABI
).send({
  gas: 4612388, from: accounts[0]
});
```

Listing 2. Signing a KYC Claim and Addition to the User's Identity SC

C. Device Initialization

In cases where the KYoT is run by the manufacturer, the device first is registered on the server before the device is sent to the customer. KYoT generates a unique key using the PUF and stores it without the user having any influence on this process. When the user eventually wants to verify his device, KYoT can compare the reproduced key derived from the new SRAM data the user will provide to the original key stored in the database by the manufacturer. The most crucial component of the device initialization by the KYoT is, therefore, the SRAM data extracted from the device. Due to the SRAM's bias toward 0 or 1, it is necessary for the device registration to use a fuzzy extractor to generate a unique key from noisy SRAM data, as well as a helper, which is needed in order to reproduce the unique key in future verification processes. Here the *fuzzy-extractor* Python library by [23] is deployed.

As pointed out in Sec. II, a proportion of the SRAM cells have no bias toward 0 or 1 and their start-up value is entirely random. This introduces noise into the SRAM reading, and consequently, the raw SRAM data cannot be used as a unique key. Therefore, KYoT error corrects the data first using a fuzzy extractor as in [23]. To use the extractor, we create a *FuzzyExtractor* object, defining the length of the input in bytes as well as the number of noisy bits we want to allow for the reproduction of the key. KYoT uses 32 Byte input strings to optimize storage costs.

D. Device Registration by the Owner

To register a device the user creates an Ethereum account and run the user registration process (*cf.* Section IV-A), which consists of several steps. Firstly, the user fills in the device registration form and assigns a device's name to differentiate between multiple devices registered. A unique identifier as generated (*cf.* Section IV-C) and shipped with the device

when purchased is assigned. This device id is used as the primary key during the creation on the database and enables the retrieval later. This process enables the self sovereign identity management of IoT devices by users as the owners.

Secondly, the identity SC is deployed to represent this device on the BC. KYoT deploys the same base SC [2] `Identity.sol` used for the user's SC to the BC. Thirdly, a user SC (*cf.* Section IV-A) is created and the user's identity is verified by adding a KYC claim to the SC. A claim key is added, too, such that the user SC can issue claims. KYoT utilizes this functionality to issue ownership claims to devices, which effectively links devices to the user and leads to verified instances of involved components represented on the BC, once devices are also verified by a KYD claim (*cf.* Listing 3).

```
// Creates a signature for the device id.
const data = this.web3.utils.asciiToHex(
  [device.value.id, 'verified'].join(' ') );
const hashedData = this.web3.utils.soliditySha3(
  result.options.address, 9, data );
const signature = await this.web3.eth.sign(
  hashedData, accounts[0]);
// Creates an ownership claim from the signature.
const user = await this.userService.get();
const addClaimABI = result.methods.addClaim(
  9, 1, user.contract, signature, data, ''
).encodeABI();
// Adds the ownership claim to the identity smart
    contract.
await result.methods.execute(
  result.options.address, 0, addClaimABI,
).send(
  { gas: 4612388, from: accounts[0] } );
```

Listing 3. Signing of an Ownership Claim and Adding it to the Device's Identity SC

KYoT uses the device's unique identifier in the signed message and indicates the claim to be an ownership claim ($claimType = 9$). The steps of adding the claim differ only in the information given on the issuer, since the user's identity SC will be given here and no URL is provided. Once these transactions are executed, the device will be represented on the BC by its proxy SC, which is also linked to the user's proxy SC through the ownership claim. In order to keep track of which devices have been registered and belong to which user, KYoT also updates the data stored in the device table of the KYD database. Thus, the device registration form along with the user's account address and the identity SC's address deployed are sent to the KYD server through the *user registration* endpoint of the REST API.

E. Device Verification

Devices registered by the user are shown on the main page, where the user can view the device's details. As long as the device is not verified by KYoT (it does not have a signed KYD claim added to its SC), it will be marked as "not verified". In this case, a button to verify the device is also displayed. To verify the device, the user will have to provide the SRAM data collected from his device. Thus, the user will be directed to detailed instructions of the process of obtaining the required data to move forward through the following steps:

1) Setting up the Arduino create editor
2) Downloading the data extraction sketch
3) Uploading the sketch to the device
4) Copying the SRAM data from the serial monitor

Once the verification details are submitted, the PUF data are sent to the KYD server alongside the device's unique identifier.

F. Identity Smart SC

The identity SC is based on ERC 734 and ERC 735 (*cf.* Section II-A). The implementation is an adjusted version of the Fractal's design [11], [19]. The implementation of the interface handles keys on the SC and represents physical entities on the BC. The ERC-based identity SCs facilitate the verification of any claim, like identity claims. While the SCs implemented are partially presented above, the source code are available at [3] and the front-end as well as detailed specifications of the KYoT source code are provided in [18].

V. Discussion of the KYoT Approach

KYoT exploits SRAM-based PUFs for self-sovereign identity provision of IoT devices using SCs. Thus, discussions are run with respect to security, privacy, and trust.

A. Security and Privacy

In an attack scenario, where a device owner wants to exploit the system, it is challenging to keep the platform secure. *E.g.*, the device owner can read out values of the entire SRAM chip, and no matter what challenge is set, as long as it is based on that SRAM, the user will know the correct response even if he is registering an imposter device. This is due to the principle of any SRAM-based PUF implementation. Such problems mainly arise from device owners having physical access to the device. With a straightforward implementation of just asking the user to use the Arduino sketch to read out the SRAM data, the device owner could return any value he considers suitable, regardless of what the sketch emits.

While a preliminary solution splitting up the registration and verification process among multiple actors, as done for KYoT, helps, it is not sufficient to guard in full against a malicious device owner. Further, device owners are verified by KYoT's KYC provider, since their BC representations are linked to their real-world identities. Therefore, malicious device owners can be held legally accountable. However, regulations are still not fully clear in the context of BCs yet. Additionally, KYoT's security depends on the security of the MetaMask log-in. Interactions with the BC, requiring the user to spent money, always have to be approved through that MetaMask extension.

To address privacy concerns, no sensitive data is published on the public BC by KYoT. Even though the retrieval of all devices registered is achieved by filtering the device list by the user's account address, the only data stored on the KYD platform being sensitive and not published on the BC is the PUF key and the user's registration data. The PUF key never leaves the KYoT's server. To protect user data collected via the KYC processes, a preferably standalone, distributed, and decoupled KYC system can protect access points better to retrieve users' data securely. The data added to signatures is intentionally kept to a minimum and never includes any data

that could be regarded as sensitive. This way, only the platform has access to the proof provided during the verification process of any entity.

B. Blockchain Integration and Trust

Generally, BC-based systems are not intended to be dependent on third parties to provide trust. In this regard, the motivation for the integration of a BC in KYoT is to remove the requirement of a trusted third party in payment transactions. However, removing the requirement for trust in KYoT would require the entire KYC and KYD process to run on the BC. That way, the verification can be handled by the protocol described in SCs and it is guaranteed to be executed identically for each device through the consensus mechanism of the BC. However, the verification process of KYoT is computationally too complex to be run in a SC and would result in high mining fees. Moreover, sensitive data would have to be sent to the SC to verify a device, which would be accessible to anyone violating privacy. Thus, the need for trust in a third-party KYD platform remains.

VI. Summary and Conclusions

This work presented the design and implementation of KYoT, a multidimensional platform for a self-sovereign identification of IoT devices. KYoT utilizes the ERC 734 and 735 standards of the Ethereum BC for IoT identification. Therefore, KYoT *(a)* enables trusted users on the BC to issue KYD claims themselves and *(b)* includes an identity proxy in the form of a SC. The generation of per-device unique keys is achieved via an SRAM-based PUF. To the best of the authors' knowledge, this is the first implementation combining a PUF-based device identification with the issuing of ERC 734/735-based claims. Evaluation results indicate that trust to device owners is unavoidable as far as a fully automated and distributed key extraction mechanism is not available via SRAM PUFs. Moreover, it is foreseeable that decoupling the KYC part from KYoT enhances the user's privacy even further.

Although more evaluations of KYoT in real world are desirable, it can be concluded that SRAM-based PUFs and their utilization in integration with identity standards have proven by this prototypical implementation a viable approach. It is highly recommended for a practical system in operation to utilize *(a)* a stronger PUF design, which goes beyond SRAMs, *(b)* periodical PUF generations without any user interactions, and *(c)* a mechanism independent of devices' power cycles. The key remains in preventing malicious parties from predicting the PUF output, otherwise, it would be impossible to ensure a fully secure authentication, since users have physical access to their devices.

VII. Acknowledgements

This paper was partially supported by (a) the University of Zürich (UZH), Switzerland and (b) the European Union Horizon 2020 Research and Innovation Program under grant agreement No. 830927, namely the Concordia Project. The authors would like to express many thanks to T.V. Prabhakar and Vaidya Girish Bhagwa, Indian Institute of Science.

References

[1] "Arduino Boards & Modules," https://store.arduino.cc/arduino-genuino/boards-modules, Last visit: October 15, 2020.

[2] "KYD Service," https://github.com/bjeffr/kyd_service.git, Last visit: October 15, 2020.

[3] "KYD Web App," https://github.com/bjeffr/kyd-gui.git, Last visit: October 15, 2020.

[4] "Memory," https://www.arduino.cc/en/tutorial/memory, Last visit: October 15, 2020.

[5] "MetaMask," https://metamask.io/, Last visit: October 15, 2020.

[6] E. Alliance, "ERC-725 Ethereum Identity Standard," https://erc725alliance.org/, Last visit: October 15, 2020.

[7] A. Babaei and G. Schiele, "Physical Unclonable Functions in The Internet of Things: State of The Art and Open Challenges," *Sensors*, Vol. 19, No. 14, p. 3208, 2019.

[8] T. Bocek and B. Stiller, "Smart Contracts - Blockchains in the Wings," in *Digital Marketplaces Unleashed*, C. Linnhoff-Popien, R. Schneider, and M. Zaddach, Eds. Germany: Springer, January 2017, pp. 169–184.

[9] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A PUF-Based Secure Communication Protocol for IoT," in *ACM Transactions on Embedded Computing Systems*, Vol. 16, No. 3, July 2017, p. 1–25.

[10] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," *SIAM Journal on Computing*, Vol. 38, No. 1, p. 97–139, 2008.

[11] Fractal, "trustfractal/erc725," https://github.com/trustfractal/erc725, Last visit: October 15, 2020.

[12] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *9th ACM Conference on Computer and Communications Security (CCS2002)*, Washington, DC, USA, November 2002, p. 148–160. [Online]. Available: https://doi.org/10.1145/586110.586132

[13] U. Guin, P. Cui, and A. Skjellum, "Ensuring Proof-of-Authenticity of IoT Edge Devices Using Blockchain Technology," in *2018 IEEE International Conference on Internet of Things (iThings)*, Halifax, NS, Canada, July 2018, pp. 1042–1049.

[14] D. Holcomb, W. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," in *IEEE Transactions on Computers*, Vol. 58, No. 9, September 2009, p. 1198–1210.

[15] U. Javaid, M. N. Aman, and B. Sikdar, "BlockPro: Blockchain Based Data Provenance and Integrity for Secure IoT Environments," in *1st Workshop on Blockchain-Enabled Networked Sensor Systems (SenSys 2018)*, Shenzhen, China, 2018, p. 13–18. [Online]. Available: https://doi.org/10.1145/3282278.3282281

[16] L. Negka, G. Gketsios, N. A. Anagnostopoulos, G. Spathoulas, A. Kakarountas, and S. Katzenbeisser, "Employing Blockchain and Physical Unclonable Functions for Counterfeit IoT Devices Detection," in *International Conference on Omni-Layer Intelligent Systems (COINS 2019)*, Crete, Greece, May 2019, p. 172–178. [Online]. Available: https://doi.org/10.1145/3312614.3312650

[17] M. Platonov, J. Hlavác, and R. Lórencz, "Using Power-Up SRAM State of Atmel ATmega1284P Microcontrollers as Physical Unclonable Function for Key Generation and Chip Identification," in *Information Security Journal: A Global Perspective*, Vol. 22, No. 5-6, February 2013, p. 244–250.

[18] S. Rafati Niya, B. Jeffrey, and B. Stiller, "A Blockchain-based Platform for Self-sovereign IoT Identification," IFI-TecReport No. 2020.04, Zürich, Switzerland, Tech. Rep., September 2020. [Online]. Available: https://owncloud.csg.uzh.ch/index.php/s/cp4JrDMYsBATaXX

[19] J. Santos, "First Impressions with ERC 725 and ERC 735 - Identity and Claims," https://hackernoon.com/first-impressions-with-erc-725-and-erc-735-identity-and-claims-4a87ff2509c9, June 2018, Last visit: October 15, 2020.

[20] F. Vogelsteller, "ERC: Claim Holder· Issue #735· ethereum/EIPs," https://github.com/ethereum/EIPs/issues/735, October 2017.

[21] ——, "ERC: Key Manager· Issue #734· ethereum/EIPs," https://github.com/ethereum/EIPs/issues/734, October 2017.

[22] H. Watanabe, "Can Blockchain Protect Internet-of-Things?" https://arxiv.org/abs/1807.06357v2, July 2018, Last visit: October 15, 2020.

[23] C. Yagemann, "Fuzzy-extractor," https://pypi.org/project/fuzzy-extractor/, Last visit: October 15, 2020.